UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/746,523	12/21/2000	Donald F. Hooper	10559-269001 / P9028	1036

20985 7590 04/20/2004

FISH & RICHARDSON, PC
12390 EL CAMINO REAL
SAN DIEGO, CA 92130-2081

EXAMINER	
VO, TED T	

ART UNIT	PAPER NUMBER
2122	

DATE MAILED: 04/20/2004

#10

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No. 09/746,523	Applicant(s) HOOPER ET AL.	
	Examiner Ted T. Vo	Art Unit 2122	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *14 January 2004*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-22* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-22* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. This action is in response to the amendment filed on 1/14/2004.

Claims 1, 4, 6, 7, 8, 16, 19 are amended.

Claims 1-22 are pending in the application.

Response to Arguments

2. With regard to Applicants' amendment to Claim 19 previously rejected under 35 USC 112 second paragraph; the rejection of Claim 19 under 35 USC 112 second paragraph is withdrawn.

3. Claims 1-21, are rejected under 35 U.S.C. 102(a) as being anticipated by Intel IXP 1200 Manual, and Claim 22 rejected under 35 U.S.C. 103(a) as being unpatentable over Intel IXP 1200 Manual, where Applicants' argument given in their Remarks (page 6) to the rejection of is that the IXP 1200 Manual neither describes nor suggests at least the feature recited in Independent Claims 1, 8, 16, and 22:

"*inserting a segment of executable code into an unused section of a target microengine's microstore in response to a first context swap of one of a plurality of hardware-supported execution threads of a program executing in the target microengine*", where Applicants contend that the IXP 1200 manual merely discloses the insertion of breakpoint instructions and not a segment of executable code (re: Remarks: page 6, started from line 12 to the end of the page).

Examiner disagrees: When the IXP 1200 manual refers "instruction", it has means of executable code. For example, page 4-48, at step 2: "breakpoint routine placed in unused code space". It is known that "code space" like given in the Example 4-5 (page 4-49) stores executable code.

Furthermore, see page 4-48: steps 1-8, and particularly, referring to:

- Step 1: "*The strong ARM Core saves the address and the instruction where the breakpoint should occur*" (Examiner interprets "performing insertion");

- Step 3: "*If the user choose to break only of a certain context hits the breakpoints*, (examiner interprets execution) *the first instruction of the breakpoint routine should be a "br!=ctx[n] to step 7*".

- Step 7: "....*When a Microengine thread is woken, it will start executing at the instruction following the step 7. This where the original instruction removed from the program step 1*".

It shows the IXP 1200 manual clearly to indicate the instructions that are inserted into the microengine is ***executable*** by such an engine.

Specifically, in the Example 4-5 (page 4-49), it shows "**break_routine**" (executable code) lies in an unused portion of Control Store of a Microengine is such code. Furthermore, the teaching of Control Store and Microengine is corresponding to the teaching in the Specification (re Spec: page 11, lines 2-12), "*hop code 516 is inserted into an unused section of the target thread's allocation of micro-store*".

Therefore, the IXP 1200 manual indeed discloses at least the feature recited in Independent Claims 1, 8, 16, and 22 and the claim limitation as quoted above.

4. Claims 1-15 are rejected under 35 U.S.C. 102(b) as being anticipated by Xu et al., "Dynamic Instrumentation of Thread Applications", Claims 16-21 are rejected under 35 U.S.C. 103(a) as being unpatentable over Xu, in view of Jacobson et al., "Asynchronous Microengines for Efficient High Level Control", and Claim 22 is rejected under 35 U.S.C. 103(a) as being unpatentable over Xu,

where Applicants' arguments given in their Remarks (re: Remarks: page 7) to the rejection of Claims 1-15 under 35 U.S.C. 102(b) as being anticipated by Xu et al., "Dynamic Instrumentation of Thread Applications", 1999 ACM, and Claim 22 as being unpatentable over Xu, where Applicants merely argue that Xu neither describes nor suggest at the quoted feature above, where Applicants merely contend that XU discloses a patch to base-trampoline (re: Remarks: page 7, lines 7-12).

Examiner disagrees: As noted in the teaching of IXP 1200 manual, the manual disclose an executable routine inserted into an unused space of a control store of a microengine, Xu discloses the a method of debugging code that is in the similar manner as of the IXP 1200 manual, where the executable code is an inserted patch code (Examiner's interpretation: "executable code) into to a base-trampoline (examiner's interpretation: "target microengine"). In Xu's case, the terminology, "base-trampoline" and

"target micro engine" is different; however, the method and the purpose for insertion of patch code, disclosed by Xu, is not distinguishable from the IXP 1200 manual.

Applicants' arguments to the teaching of Xu is simply on terminology, patch code (Examiner's interpretation: "executable code") and base-trampoline (Examiner's interpretation: "target microengine"), but Applicants ignore functionality of debugging disclosed by Xu. Therefore, the argument is not persuasive.

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for a patent.
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

6. Claims 1-21 are rejected under 35 U.S.C. 102(a) as being anticipated by Hardware Reference Manual, "Intel™ IXP1200 Network Processor Hardware Reference Manual", appeared in http:// www.hcs.ufl.edu/, version 1.0, June 2000.

Given the broadest reasonable interpretation of followed claims in light of the specification.

As per claim 1: The manual (Hardware reference manual) discloses,

"A method of debugging code that executes in a multithreaded processor having a plurality of microengines (see page 4-47, section 4.17, Debugging Support) ***comprising:***

inserting a segment of executable code into an unused section of a target microengine's microstore (see page 4-48, section 3.17.3: Breakpoints, lines 1-3, 'insert and remove breakpoints...', and indent 2, '... placed in unused code space..') ***in response to a first context swap of one of a plurality of hardware-supported execution threads of a program executing in the target microengine*** (see page 4-47, second and third paragraphs of section 4.17.2, particularly, 'after context switch point' in the last 2 lines);

executing the segment of executable code (see page 4-48, section 4.17.3, indents 6 and 7; and see page 4-24, first paragraph of section 4.10, '...a context switch occurs and another program begins executing.'); ***and***

resuming execution of the program in response to a second context swap (see page 4-48, section 4.17.3, indent 8, 'A branch back to the address...,).

As per claim 2: The manual discloses "***inserting comprises: saving program counters associated with the plurality of hardware-supported execution threads*** (see page 4-48, section 3.17.3: Breakpoints, indent 1, '... saves the address...');

modifying the target microengine's program counters to jump to a start of the segment of executable code (see page 4-49, Example 4-5, 'BR' and 'break_routine'); ***and***

appending the saved program counters to an end of the segment of executable code (see page 4-49, Example 4-5, (to breakpoint_point +1) after 'BR').

As per claim 3: The manual discloses, "***inserting further comprises receiving a user request to execute the segment of executable code***" (see page 4-25, section 4.10.1, 'Explicitly requested events are requested by the programmer via program instructions', and see page 4-47, section 4.17.2, third (or last) paragraph, 'user to hop').

As per claim 4: The manual discloses, "***wherein executing further comprises examining states of the hardware supported execution threads at the second context swap***" (see page 4-47, section 4.17.2, third (or last) paragraph, 'execution progress is monitored after a context switch point')

As per claim 5: Intel manual discloses, "***wherein resuming further comprises removing the segment of executable code from the microstore***" (see page 4-48, section 4.17.3, Breakpoints, first paragraph, 'dynamically insert or remove breakpoint into the Control Store')

As per claim 6: The manual disclose interrupt service (see page 4-48, section 4.17.3, Breakpoints, paragraph '*note:*', between indents 5 and 6, 'The StrongARM interrupt service routine'), where interrupt service is known for saving resumed addresses of execution for interrupting service routines). This teaching covers claim limitation: "***The method of claim 1 wherein resuming comprises restoring program counters of the plurality of hardware supported execution threads that have not executed***".

As per claim 7: The manual discloses, "***The method of claim 1 wherein the segment of executable code resides in a library of executable code segments residing in the processor***" (see page 4-47, section 4.17, first paragraph, 'Workbench and debug libraries').

As per claims 8: The manual discloses, "***A method of debugging software that executes in a multithreaded processor having a plurality of microengines comprising:***

pausing program execution in a plurality of threads of execution within a target microengine (see page 4-47, section 4.17.2, first paragraph, '..the running and Paused states..').

inserting a segment of executable code into an unused section of the target microengine's microstore (see page 4-48, section 3.17.3, Breakpoints, lines 1-3, 'insert and remove breakpoints...', and indent 2, '... placed in unused code space..');

executing the segment of executable code in the target microengine (see page 4-47, section 4.17.2, second and third paragraph, particularly, 'after context switch point' in the last 2 lines)*;* ***and resuming program execution in the target microengine"*** (see page 4-48, section 4.17.3, indent 8, 'A branch back to the address...,).

As per claim 9: The manual discloses, "**The method of claim B wherein pausing is in response to a user command to pause**" (see page 4-47, section 4.17.2, second paragraph, 'The user can manually place the Microengine into a paused state...')

As per claim 10: HRM discloses, "***The instruction of claim 8 wherein the user command to pause further comprises selecting the target microengine from one of the plurality of microengines***" (see page 4-49, section 4.17.4, first paragraph, 'When a Microengine is placed into the Stopped or Paused state...')

As per claim 11: The manual discloses, "***The method of claim 10 wherein the user command to pause further comprises selecting the segment of executable code***" (see page 4-47, third paragraph of section 4.17.2, (the last paragraph of the page), 'it is possible for the user to hop the execution..').

As per claim 12: The manual discloses, "***The method of claim 8 wherein pausing further comprises determining when one of the plurality of threads of execution context swaps***" (see page 4-47, section 4.17.1, third paragraph, 'reading the current PC for each of the contexts').

As per claim 13: The manual discloses, "***The method of claim 8 wherein inserting further comprises: modifying a program counter of the paused program to point to the segment of executable code; and modifying a program counter at an end of the segment of executable code to point to the paused program***" (see page 4-49, Example 4-5).

As per claim 14: The manual discloses, "***The method of claim 8 wherein the segment of executable code causes the target microengine to write to specific registers***" (see page 4-48, section 4.17.3, indent 7, '...writing to the CTX_ENABLES...').

As per claim 15: The manual discloses, "***The method of claim 14 wherein the specific registers are examined by a user during execution of the segment of executable code***" (see page 4-49, first paragraph of section 4.17.4, 'The ALU_OUTPUT CSR can be read...').

As per claim 16: Regarding, "***A processor that can execute multiple contexts comprising: a register stack; a program counter for each executing context; an arithmetic logic unit coupled to the register stack and a program control store that stores a breakpoint command*** (see page 4.1, section 4.1: Overview) ***that causes the processor to***":

pause program execution in a context in the processor; insert a segment of executable code into a used section of a microstore associated with the context; execute the segment of executable code; and resume program execution":

The claim limitation has the functionality corresponding to the functionality of claim 8.

Claim 16 has the claim functionality corresponding to the functionality of claim 8; therefore it is rejected in the same reason as set forth in connecting to the rejection of claim 8.

As per claim 17: The manual discloses, "***The processor of claim 16 wherein program execution is paused by disabling a processor enable bit***", (see page 4-14, section 4.6.4, second paragraph, '...writing to the CTX_ENABLES register').

As per claim 18: The manual discloses, "***The processor of claim 16 wherein the segment is inserted in response to a user request received through a remote user interface connected to the processor***" (see page 4-48, section 4.17.3, indent 3, 'If the user chooses').

As per claim 19: The manual discloses, "***The processor of claim 16 wherein an end of the segment points to a program counter of the program***" (see page 4-49, Example 4-5, particularly, 'break_point' in the routine).

As per claim 20: The manual discloses, "***The processor of claim 16 wherein the segment examines states of execution of the contexts***" (see page 4-49, section 4.17.4, first paragraph, 'The ALU_OUTPUT CSR can be read...').

As per claim 21: The manual discloses, "***The processor of claim 16 wherein program execution is resumed by enabling a processor enable bit***" (see page 4-13, section 4.6.3, '...enabled via CTX_ENABLES registers').

7. Claims 1-15 are rejected under 35 U.S.C. 102(b) as being anticipated by Xu et al., "Dynamic Instrumentation of Thread Applications", 1999 ACM.

Given the broadest reasonable interpretation of followed claims in light of the specification.

As per claim 1: Xu discloses,

"A method of debugging code that executes in a multithreaded processor (see page 1, second paragraph, referring to all four bullets) ***having a plurality of microengines*** (see page 1, fist column, third paragraph, 'multiprocessor Sun UltraSPARC') ***comprising:***

inserting a segment of executable code (See page 2, second column, first paragraph, 'patches a jump to a base-trampoline'; and see page 4, second column, section 3.4, first paragraph; 'past the point in which it was inserted') ***into an unused section of a target microengine's microstore*** (see page 2, second column, first paragraph, 'relocates the instructions that were overwritten', see page 3, figure 3, Mini Trampoline; or see, page 3, second column, second paragraph, 'we add code to the base-trampoline....to detect the first time a thread executes instrumentation code');

in response to a first context swap of one of a plurality of hardware-supported execution threads of a program executing in the target microengine (see page 3, figure 3, referring to the exit arrow at the function foo, and the destination at: compute C/T Addr)

executing the segment of executable code (see page 3, figure 3, Mini Trampoline; and see page 4, second column, section 3.4, third paragraph, 'Our solution...to execute the code on behalf of the particular thread that needs to run the inferior RPC'); ***and***

resuming execution of the program in response to a second context swap (see page 3, figure 3, the return arrow from Mini Trampoline, and the reentry arrow after function foo, or see page 4, second column, second paragraph, 'every thread context switch to account for any possible thread migration').

As per claim 2: Xu discloses ***"inserting comprises: saving program counters associated with the plurality of hardware-supported execution threads*** (see page 3, figure 3 'Save Regs').

modifying the target microengine's program counters to jump to a start of the segment of executable code (see page 3, figure 3 'Post Instrument'). ***and***

appending the saved program counters to an end of the segment of executable code (see page 3, figure 3, referring to a blank area in Mini Trampoline connected with the return arrow; and see description in section 3.4 second column, third paragraph (page 4)).

As per claim 3: Xu discloses, ***"inserting further comprises receiving a user request***

to execute the segment of executable code" (see page 4, second column, section 3.4, first paragraph, 'we use inferior to trigger the entry instrumentation').

As per claim 4: Xu discloses, "***wherein executing further comprises examining states of the hardware supported execution threads at the second context swap***" (see page 4, first column, third paragraph, referring to 'add extra information to the lock, counter, and timer structures to effect communication between interleaving instrumentations').

As per claim 5: Xu discloses, "***wherein resuming further comprises removing the segment of executable code from the microstore***" (see page 2, first column, section 2, first paragraph, '..removed from a running application', and see page 3, second column, second paragraph, particularly, '...to ensure that no thread is executing the instrumentation code when we try to remove it').

As per claim 6: Xu describes 'restore register' located after 'Pre instrument' or 'Post instrument' in the Base Trampoline (see page 3, figure 3). This teaching cover the limitation: "***The method of claim 1 wherein resuming comprises restoring program counters of the plurality of hardware supported execution threads that have not executed***" (see page 3, figure 3, 'Base Trampoline').

As per claim 7: Xu discloses, "***The method of claim 1 wherein the segment of executable code resides in a library of executable code segments residing in the processor***" (see page 4, second column, section 3.4, fourth paragraph, 'RPCs in a shared memory segment accessible by both Paradyn and the application').

As per claims 8: Xu discloses, "***A method of debugging software that executes in a multithreaded processor having a plurality of microengines comprising:***

pausing program execution in a plurality of threads of execution within a target microengine (see page 4, second column, section 3.4, second paragraph, 'Paradyn implements inferior RPC by pausing the application,');

inserting a segment of executable code (See page 2, second column, first paragraph, 'patches a jump to a *base-trampoline* ', and see page 4, second column, section 3.4, first paragraph, 'past the point in which it was inserted') ***into an unused section of the target microengine's microstore*** (see page 2, second column, first paragraph, 'relocates the instructions that were overwritten', or see page 3, figure 3,

Mini Trampoline; [Examiner interprets 'target microengine's microstore' as the location that store the code of Mini Trampoline], or see, page 3, second column, second paragraph, 'we add code to the base-trampoline to detect the first time a thread executes instrumentation code' [examiner interprets 'unused section' as an area that stores the thread executes instrumentation code]);

executing the segment of executable code in the target microengine (see figure 3, Mini Trampoline; and see page 4, second column, section 3.4, third paragraph, 'Our solution...to execute the code on behalf of the particular thread that need to run the inferior RPC'); ***and resuming program execution in the target microengine"*** (see figure 3, the return arrow from Mini Trampoline, and the reentry arrow after function foo, see page 4, second column, second paragraph, 'every thread context switch to account for any possible thread migration').

As per claim 9: Xu discloses, "**The method of claim 8 wherein pausing is in response to a user command to pause**" (see page 4, second column, section 3.4, second paragraph, 'Paradyn implements inferior RPC', and referring to page 1, second column, all fourth bullets).

As per claim 10: Xu discloses, "***The instruction of claim 8 wherein the user command to pause further comprises selecting the target microengine from one of the plurality of microengines***" (see page 4, second column, section 3.4, third paragraph, 'passing the ID'; and 'thread Table').

As per claim 11: Xu discloses, "***The method of claim 10 wherein the user command to pause further comprises selecting the segment of executable code***" (see page 4, second column, section 3.4, second paragraph, 'and changing the program PC to the inferior RPC code to execute it').

As per claim 12: Xu adds information to the lock, counters and timer structure to effect communication between inter leaving instrumentations (see page 4, first column, third paragraph). Xu uses instrument thread context switches to account for thread context switching and migration (see page 4, second column, first paragraph). This teaching helps to determine the execution of multiple context switches and this teaching covers the claim limitation: "***The method of claim 8 wherein pausing further comprises determining when one of the plurality of threads of execution context swaps***".

As per claim 13:

Xu shows an address of function foo that is modified to cause a branch to Base trampoline (see page 3, figure 3). During the execution of Base Trampoline, the program application is not executed ('paused') (referring page 4, second column, section 3.4, second paragraph, 'pausing...'). This mechanism is further described in page 2, section 2: Paradyn Basics. This teaching covers the claim limitation: "***The method of claim 8 wherein inserting further comprises: modifying a program counter of the paused program to point to the segment of executable code".***

Xu shows an address (of function foo) that is returned after executing Base Trampoline and Mini Trampoline (page 3, figure 3). This teaching covers the claim limitation.
"and modifying a program counter at an end of the segment of executable code to point to the paused program".

As per claim 14: Xu discloses, "***The method of claim 8 wherein the segment of executable code causes the target microengine to write to specific registers***" (see page 3, first column, first paragraph, referring to, 'The column address is stored in a register')

As per claim 15: Xu discloses, "***The method of claim 14 wherein the specific registers are examined by a user during execution of the segment of executable code***" (see page 3, first column, first paragraph, 'an extra piece of code computes the counter or timer address based on the value of the register that hold the column address').

Claim Rejections - 35 USC § 103

8. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> A person shall be entitled to a patent unless –
>
> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

9. Claim 22 is rejected under 35 U.S.C. 103(a) as being unpatentable over Hardware Reference Manual, "Intel™ IXP1200 Network Processor Hardware Reference Manual", appeared in http://www.hcs.ufl.edu/, version 1.0, June 2000.

Given the broadest reasonable interpretation of followed claim in light of the specification:

As per claim 22: Regarding limitation, "***A computer program product, disposed on a computer readable medium, the product including instructions for causing a multithreaded processor having a plurality of microengines to:***

pause program execution in a plurality of threads of execution within a target microengine; insert a segment of executable code into an unused section of the target microengine's microstore; execute the segment of executable code in the target microengine; and resume program execution in the target microengine":

The claim is a product claim that is disposed on a computer medium. The claim has the functionality corresponding to the functionality method claim 8.

The manual, therefore, discloses the functionality of claim 22. This is rejected in the same reason as set forth in connecting to the rejection of claim 8 (section: (3)(**3.A**)(As per claim 8)).

The manual does not address, "***computer program product, disposed on a computer readable medium***". However, a product like a computer readable medium that stores a computer program is common in the art for implying all availability of computer technology such as read/write disk storages and operating systems.

Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to program a method that perform multithread debug as disclosed by the manual into a computer product like a computer readable medium.

The modification would be obvious because one of ordinary skill in the art would be motivated for saving an execution algorithm by taking advantage of all availabilities of computer technology such as read/write command of an operating system and disk storage for later uses or business purposes.

10. Claim 22 is rejected under 35 U.S.C. 103(a) as being unpatentable over Xu et al., "Dynamic Instrumentation of Thread Applications", 1999 ACM.

As per claim 22: Regarding limitation, "***A computer program product, disposed on a computer readable medium, the product including instructions for causing a multithreaded processor having a plurality of microengines to:***
pause program execution in a plurality of threads of execution within a target microengine; insert a segment of executable code into an unused section of the target microengine's microstore; execute the segment of executable code in the target microengine; and resume program execution in the target microengine":
The claim is a product claim that is disposed on a computer medium. The claim has the functionality corresponding to the functionality method claim 8.
Xu, therefore, discloses the functionality of claim 22. This is rejected in the same reason as set forth in connecting to the rejection of claim 8 (section: (3)(**3.B**)(As per claim 8)).
Xu does not address. "***computer program product, disposed on a computer readable medium***".
However, a product like a computer readable medium that stores a computer program is common in the art for implying all availability of computer technology such as read/write disk storage and operating system.

Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to program a method that perform multithread debug as disclosed by the manual into a computer product like a computer readable medium.
The modification would be obvious because one of ordinary skill in the art would be motivated for saving an execution algorithm by taking advantage of all availabilities of computer technology such as read/write command of an operating system and disk storage for later uses or business purposes.

11. Claims 16-21 are rejected under 35 U.S.C. 103(a) as being unpatentable over Xu et al., "Dynamic Instrumentation of Thread Applications", 1999 ACM, in view of Jacobson et al., "Asynchronous Microengines for Efficient High Level Control", 1997 IEEE.

Given the broadest reasonable interpretation of followed claim in light of the specification:

As per claim 16:

Xu discloses a multiprocessor ('***microengines***') (page 1, fist column, third paragraph, 'multiprocessor Sun UltraSPARC') that is implemented with a method to perform dynamic instrumentation of threaded applications. At runtime, the dynamic instrumentation ('***control store***') requests patching a jump ('***stores a breakpoint command*** ')(see page 2, second column, first paragraph) to another thread execution (see page 3, figure 3, Mini Trampoline) that is stored in a share memory (page 4, second column, section 3.4, fourth paragraph, referring to 'inferior segment RPCs in a share memory'). Xu's dynamic instrumentation is associated with instrument thread switches to account for thread switching and migration (see page 4, second column, first paragraph). The switch execution causes a pausing the application program (page 4, second column, section 3.4, second paragraph, referring to "pausing the application'), where the execution mechanism is shown in figure 3.

Xu discloses the dynamic instrumentation that covers claim limitation: "***a program control store*** (dynamic instrumentation) ***that stores a breakpoint command*** (see page 2, second column, first paragraph, 'patches a jump') ***that causes the processor to***:

pause program execution in a context in the processor (page 4, second column, section 3.4, second paragraph, 'pausing the application')

insert a segment of executable code into an used section (page 4, second column, section 3.4, fourth paragraph, 'share memory segment') ***of a microstore associated with the context;*** (page 3, figure 3, Mini Trampoline);

execute the segment of executable code (page 4, second column, section 3.4, second paragraph, 'and changing the program PC to the inferior RPC code to execute it' '); ***and***

resume program execution" (page 3, figure 3, referring to the return arrow after the function foo).

Xu does not address, a register stack, an arithmetic logic unit coupled to the register stack, as being configured in the limitation "***A processor that can execute multiple contexts and that comprises: a register stack; a program counter for each execution context; an arithmetic logic unit coupled to the register stack***".

Jacobson discloses basis architecture of microengines that covers the limitation, "***a register stack, an arithmetic logic unit coupled to the register stack***" (Jacobson: see page 203, section 2, and figure 1). Since context switching that cause pausing is common knowledge, it is performable by the microengine architecture as detailed by Jacobson.

Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to combine a method that uses multiprocessor to perform dynamic instrumentation as disclosed by Xu and a basis of microengine architecture of Jacobson.

The modification would be obvious because one of ordinary skill in the art would be motivated for conforming to an execution principle, where each processor has to have a set of standard elements for performing such a principle.

As per claim 17: Xu's dynamic instrumentation measures CPU performance of function foo. It pauses the execution of an application program during context switching. It uses start timer and stop timer to measure the CPU performance (Xu: page 4, second column, section 3.4, second paragraph). The switch that causes the pause and insertion of start timer code causes internal hardware to pause (disable bit) the execution of the application program. This teaching covers claim limitation: "***The processor of claim 16 wherein program execution is paused by disabling a processor enable bit***".

As per claim 18: Xu's dynamic instrumentation performed by multiprocessors discloses, "***The processor of claim 16 wherein the segment is inserted in response to a user request received through a remote user interface connected to the processor***" (Xu: see page 2, second column, first paragraph, 'runtime instrument request', and referring to page 1, second column, all fourth bullets).

As per claim 19: Xu's dynamic instrumentation performed by multiprocessors discloses, "***The processor of claim 16 wherein an end of the segment pints to a program counter of the program***" (Xu, see figure 3, referring the path that returns from Mini Trampoline to the application program).

As per claim 20: Xu's dynamic instrumentation performed by multiprocessors discloses, "***The processor of claim 16 wherein the segment examines states of execution of the contexts***" (Xu: page 6, referring the contents of table 2 and table 3).

As per claim 21: Xu's dynamic instrumentation measures CPU performance of function foo. It pauses the execution of an application program during context switching. It uses start timer and stop timer to measure the CPU performance (Xu: page 4, second column, section 3.4, second paragraph). The switch that causes the pause and insertion of stop timer code causes internal hardware to resume (enable bit) the execution of the application program. This teaching covers claim limitation: "***The processor of claim 16 wherein program execution is resumed by enabling a processor enable bit***".

Conclusion

12. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ted T. Vo whose telephone number is (703) 308-9049. The examiner can normally be reached on Monday-Friday from 8:00 AM to 5:30 PM ET. If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Dam, can be reached on (703) 305-4552.

The fax phone numbers:

(703) 872-9306 (for formal communication intended for entry);

(703) 746-5429 (for informal or draft communication, please label "PROPOSED" or "DRAFT").

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the Group receptionist whose telephone number is (703) 305-3900.

TTV

Art Unit: 2122
April 19, 2004

TUAN DAM
SUPERVISORY PATENT EXAMINER